



EXCALIBUR™

EPXA1 Development Kit

Getting Started



101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
<http://www.altera.com>

UG-EPXA1-1.3
P25-08163-00

User Guide
January 2003



Copyright © 2003 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, mask work rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.



I.S. EN ISO 9001

This user guide provides comprehensive information about getting started with the Altera® Excalibur™ EPXA1 development kit.

Table 1 shows the user guide revision history.

<i>Table 1. User Guide Revision History</i>	
Date	Description
January 2003	Minor text edits.
November 2002	Updates for Quartus® II version 2.2.
September 2002	CD name change.
August 2002	Initial release.

How to Find Information

- The Adobe Acrobat Find feature allows you to search the contents of a PDF file. Click on the binoculars icon in the top toolbar to open the **Find** dialog box.
- Bookmarks serve as an additional table of contents.
- Thumbnail icons, which provide miniature previews of each page, provide a link to the pages.
- Numerous links, shown in green text, allow you to jump to related information.

How to Contact Altera

For the most up-to-date information about Altera products, go to the Altera world-wide web site at <http://www.altera.com>.

For technical support on this product, go to <http://www.altera.com/mysupport>. For additional information about Altera products, consult the sources shown in [Table 2](#).




<i>Table 2. How to Contact Altera</i>		
Information Type	USA & Canada	All Other Locations
Technical support	http://www.altera.com/mysupport/	http://www.altera.com/mysupport/
	(800) 800-EPLD (3753) (7:00 a.m. to 5:00 p.m. Pacific Time)	(408) 544-7000 (1) (7:00 a.m. to 5:00 p.m. Pacific Time)
Product literature	http://www.altera.com	http://www.altera.com
Altera literature services	lit_req@altera.com (1)	lit_req@altera.com (1)
Non-technical customer service	(800) 767-3753	(408) 544-7000 (7:30 a.m. to 5:30 p.m. Pacific Time)
FTP site	ftp.altera.com	ftp.altera.com

Note:

(1) You can also contact your local Altera sales office or sales representative.

Typographic Conventions

The *EPXA1 Development Kit Getting Started User Guide* uses the typographic conventions shown in [Table 3](#).

<i>Table 3. Conventions</i>	
Visual Cue	Meaning
Bold Type with Initial Capital Letters	Command names, dialog box titles, checkbox options, and dialog box options are shown in bold, initial capital letters. Example: Save As dialog box.
bold type	External timing parameters, directory names, project names, disk drive names, filenames, filename extensions, and software utility names are shown in bold type. Examples: f_{MAX} , \QuartusII directory, d: drive, chiptrip.gdf file.
<i>Italic Type with Initial Capital Letters</i>	Document titles are shown in italic type with initial capital letters. Example: <i>AN 75 (High-Speed Board Design)</i> .
<i>Italic type</i>	Internal timing parameters and variables are shown in italic type. Examples: <i>t_{PIA}</i> , <i>n + 1</i> . Variable names are enclosed in angle brackets (< >) and shown in italic type. Example: < <i>file name</i> >, < <i>project name</i> >.pof file.
Initial Capital Letters	Keyboard keys and menu names are shown with initial capital letters. Examples: Delete key, the Options menu.
“Subheading Title”	References to sections within a document and titles of Quartus II. Help topics are shown in quotation marks. Example: “Configuring a FLEX 10K or FLEX 8000 Device with the BitBlaster™ Download Cable.”
Courier type	Signal and port names are shown in lowercase Courier type. Examples: data1, tdi, input. Active-low signals are denoted by suffix _n, e.g., reset_n. Anything that must be typed exactly as it appears is shown in Courier type. For example: c:\quartusII\qdesigns\tutorial\chiptrip.gdf. Also, sections of an actual file, such as a Report File, references to parts of files (e.g., the AHDL keyword SUBDESIGN), as well as logic function names (e.g., TRI) are shown in Courier.
1., 2., 3., and a., b., c.,...	Numbered steps are used in a list of items when the sequence of the items is important, such as the steps listed in a procedure.
■	Bullets are used in a list of items when the sequence of the items is not important.
v	The checkmark indicates a procedure that consists of one step only.
	The hand points to information that requires special attention.
	The angled arrow indicates you should press the Enter key.
	The feet direct you to more information on a particular topic.



Notes:

About this User Guide	iii
How to Find Information	iii
How to Contact Altera	iv
Typographic Conventions	v
Getting Started	9
Introduction	9
Software and Hardware Requirements	9
Excalibur Utilities	10
Quartus II Software	10
ADS-Lite	10
GNUPro Toolkit	10
EPXA1 Development Board	11
Preparation	11
Design Overview	12
Configure the Stripe	15
Compile the Hardware Design	16
Compile the Software Application	17
Specifying the Toolset Directory	17
Software Build Settings for ADS	18
Debug Settings for ADS	19
Release Settings for ADS	20
Software Build Settings for GNUPro	21
Debug Settings for GNUPro	22
Release Settings for GNUPro	24
Configure the Development Board	25
Boot-from-Passive-Serial Mode	25
Boot-from-Flash Mode	26
Connections and Jumper Settings	27
Altera Flash Programmer	28
Debug the Design	28
ADS AXD Debugger	28
GNUPro Insight Debugger	30
Appendix A—	
Installing the ByteBlaster Driver	33
Introduction	33
Installing the ByteBlaster Driver on a Windows NT System	33

Installing the ByteBlaster Driver on a Windows 2000 System	34
Confirming the Installation	35

Introduction

The EPXA1 development kit contains everything you need to develop complete embedded systems on a programmable chip. It contains the EPXA1 development board featuring the Excalibur™ EPXA1 device.



For more details about the EPXA1 device, refer to the *Excalibur Devices Hardware Reference Manual*.

The EPXA1 development kit also contains example reference designs. The designs demonstrate the interfaces and applications possible for system-on-a-programmable-chip (SOPC) designs and serve as platforms for creating new designs.

The design example referenced in this guide is a simple implementation of a Hello World program that can be used as a template for your own applications. The Hello World program is commonly one of the first applications an engineer writes when developing software code for a new processor: successfully building and executing it on the processor validates the tool chain for developing embedded software. For the Excalibur devices, a successful build and execution also validates the hardware configuration of the device and the programmable logic.

The Altera version of the Hello World program prints a message to a terminal window and scrolls the LEDs on the EPXA1 development board. The hardware and software source code is provided, which may be useful as the basis for developing additional applications.

Software and Hardware Requirements

The following list summarizes the hardware and software development tools needed to build and run the example design in the Windows NT/2000 environment:

- The Excalibur utilities
- The Quartus® II software, version 2.2
- Either of the following:
 - Red Hat GNUPro Toolkit for ARM
 - ARM Developer Suite for Altera (ADS-Lite) software version 1.1
- EPXA1 development board



The listed tools may not be included in the development kit, although they are required to run this example design; refer to the development kit contents list for the items included in the development kit.

Excalibur Utilities

The Excalibur utilities are applications required to start using the EPXA1 development board. The utilities included allow you to create programming files, set up the programming hardware, and download applications to flash memory. The utilities are installed automatically when you install the Quartus II software. If you do not need to modify the hardware portions of the design and do not require Quartus II, you can install the utilities separately via the standalone installer available on the SOPC Builder CD-ROM.

Quartus II Software

The Quartus II development software provides a comprehensive environment for SOPC design. The Excalibur MegaWizard® Plug-In, included with the Quartus II software, enables quick and intuitive setup and customization of the embedded processor in the EPXA1 device. To create new projects or to modify embedded hardware in an existing EPXA1 project, the Quartus II development software is required.

GNUPro Toolkit

GNUPro Toolkit is a complete solution for C and C++ development for the ARM processors, including a compiler, a debugger, binary utilities, libraries, and other tools. The GNUPro Toolkit is included with the Quartus II software.



Refer to the *GNUPro Toolkit User's Guide for Altera for ARM and ARM/Thumb® Development* for more information.

ADS-Lite

ADS-Lite is a collection of embedded software development tools for Excalibur devices. The tools include a C/C++ compiler, assembler, linker, debugger, and various utilities. To create new embedded software applications or to modify embedded software applications for an existing EPXA1 project, the Altera ADS-Lite software (or another compiler that supports the ARM922T processor) is required. You can purchase ADS-Lite from your Altera representative.

EPXA1 Development Board

The EPXA1 development board is a desktop development system. It provides a hardware platform to begin developing embedded systems. The development board provides a flexible and powerful environment for debugging your own designs.



For more details about the EPXA1 development board, refer to the *Excalibur EPXA1 Development Board Hardware Reference Manual*.

Preparation

When you receive your EPXA1 development kit, unpack the box and ensure that all the expected contents are present. At this point, you can make all the connections to the board, turn it on, and ensure it is working properly. The steps below explain how to do this in greater detail:


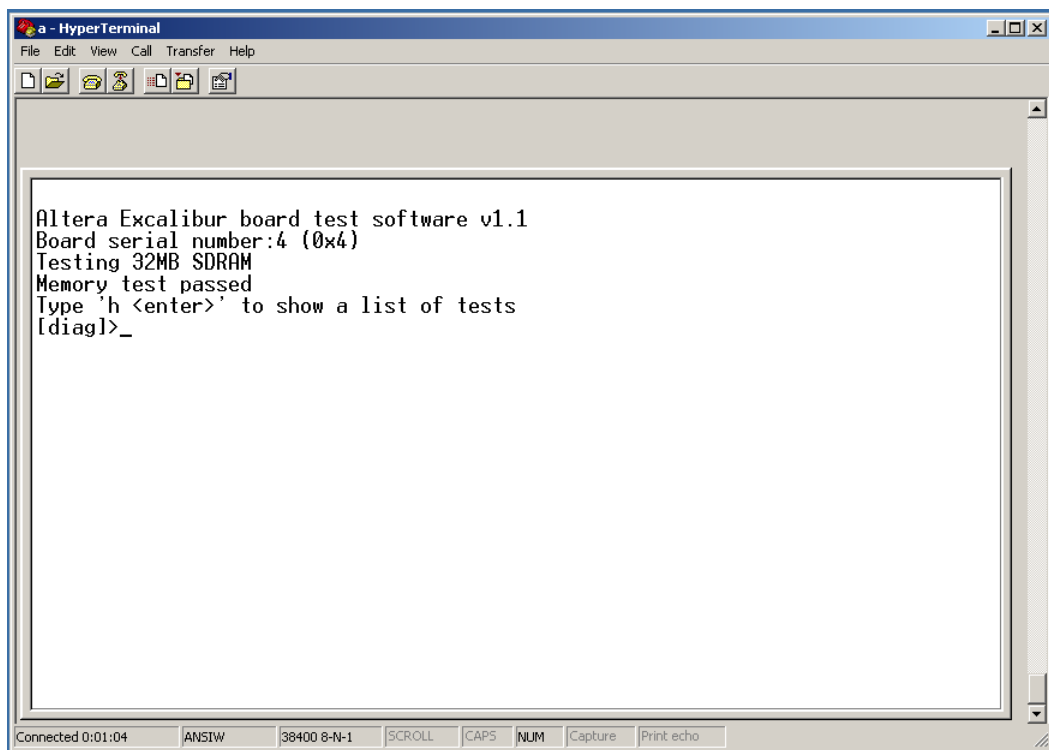
1. Take the following items out of the box:
 - Power supply
 - Appropriate power cord for your outlet type
 - RS-232 null-modem serial cable
 - EPXA1 development board
2. Connect one end of the RS-232 serial cable into the DB9 connector, P2, on the EPXA1 board, and the other into an available serial port on your PC.
3. Start up HyperTerminal and configure it for 38400 baud rate, 8 data bits, 1 stop bit, no parity, and no flow-control.
4. Configure the ASCII setup of HyperTerminal by clicking **File > Properties > Settings > ASCII Setup**. Turn on **Send line ends with line feeds** and **Echo typed characters locally**.
5. Plug the barrel connector of the power supply into the receptacle on the board
6. Plug one end of the power cord into the power supply and other to the outlet.
7. The EPXA1 Board Diagnostics program should appear in the HyperTerminal window, allowing you to carry out board diagnostic tests that exercise and test the Ethernet, SDRAM, and UART interfaces. Type **h**  to see a listing of the tests available to run. See [Figure 1 on page 12](#) for a screenshot of the Hyper Terminal window.

Figure 1. HyperTerminal Window

If you need to re-load the **EPXA1 Board Diagnostics** program onto the board, the design is located on the *Excalibur Device Applications* CD-ROM.

Design Overview

The Hello World project includes a simple PLD design and an embedded software application.

The PLD design consists of a memory-mapped slave peripheral that accepts 8-bit data. The data lights up corresponding LEDs on the EPXA1 development board. The embedded software program writes a data pattern to the slave peripheral to scroll the LEDs and also print the data pattern to a terminal window.

In this example, the compiler library functions **printf()** and **scanf()** interface to a terminal window by utilizing the **fputc** (send a character) and **fgetc** (receive a character) functions to use the EPXA1 stripe UART. **Printf()** is used for displaying characters to a terminal window and **scanf()** is used to receive characters from a terminal keyboard.

The Hello World project has the following directory structure:

```
\ads
  \Debug
  \Release
  \software
\gnu
  \Debug
  \Release
  \software
\common
\rtl
```

The following general points apply to the directories:

- The **ads** directory contains the Quartus project files, top-level hardware design, and software startup files that are specific to the ADS tools.
- The **gnu** directory contains the Quartus project files, top-level hardware design, and software startup files that are specific to the GNU tools.
- The **common** directory contains C programs that are used by both the ADS and GNU tools.
- The **rtl** directory contains Verilog HDL files of the slave peripheral

Table 1 lists the design files in the archive.

Table 1. Hello World Design Files	
File	Description
<ads or gnu>\arm_top.bdf	Block diagram file for top-level design.
<ads or gnu>\hello.csf	Compiler settings file for top-level design that stores chips definitions, device options, compilation type, etc.
<ads or gnu>\hello.psf	Project settings file that stores working directory name, relative hierarchical assignments, device assignments, etc.
<ads or gnu>\hello.quartus	Project configuration file that stores input filenames and compiler settings files.
<ads or gnu>\hello.sbd	System build descriptor file that stores information on interconnections between modules and how they should be configured.
<ads or gnu>\prog_hw.bat	Batch file for programming hardware and software image into the flash of EPXA1 evaluation board.
<ads or gnu>\stripe.v	Verilog HDL instantiation of embedded stripe.
<ads or gnu>\stripe.bsf	Block symbol of embedded stripe.
ads\Debug.fsf	Software build settings file for ADS that stores compiler options with debug information and no code optimization.
ads\Release.fsf	Software build settings file for ADS that stores compiler options without debug information and with code optimization.
gnu\Debug.fsf	Software build settings file for GNU that stores compiler options with debug information and no code optimization.
gnu\Release.fsf	Software build settings file for GNU that stores compiler options without debug information and with code optimization.
ads\software\armc_startup.s	Assembly file for ADS which initializes the stack pointers, sets up the interrupt handlers, enables the instruction and data caches, sets up the MMU, and finally jumps to the main program.
ads\software\retarget.c	C file that implements functions necessary to link with the ARM C libraries.
gnu\software\crt0.s	Assembly file for GNU which initializes the stack pointers, sets up the interrupt handlers, enables the instruction and data caches, sets up the MMU, and finally jumps to the main program.
gnu\software\epxa1.c	C file for GNU which provides the functions necessary to run the C runtime environment.
gnu\software\armelf.x	Linker script file for GNU.
common\main.c	C file that scrolls the LEDs and sends messages to a terminal window.
common\uartcomm.c	C file that implements the UART I/O functions to enable the printf function.
common\irq.c	C file that initializes the interrupt controller, first-level IRQ and FIQ handlers.
common\exceptions.c	C file that handles exceptions.
rtl\single_transaction_slave.v	Verilog HDL file of slave peripheral.
rtl\regfile	Verilog HDL register file.

Configure the Stripe

The example design has been pre-built. To become familiar with the process, or to modify it for your own designs, perform the following steps to configure the stripe:

1. Run the Quartus II software.
2. Choose **Open Project** (File menu) and select the **hello.quartus** project file from either the **ads** or **gnu** directory, depending on the compiler you are using.
3. Choose **Open** (File menu) and select **arm_top.bdf** to open the block diagram of the top-level design.
4. Double-click on the **stripe** module to invoke the MegaWizard® Plug-In.
5. Specify the following settings and click **Next**:
 - **Excalibur family**: Excalibur_ARM
 - **Available device**: EPXA1
 - **Byte order**: Little endian
 - **UART** (under **Reserve pins**): turn on
 - **SDRAM** (under **Reserve pins**): turn on



The EPXA1 UART I/O pins are enabled, because the design sends a message to a terminal window through this peripheral. The SDRAM I/O pins are enabled, because the software portion of the design will be stored in SDRAM.

- Under **Reset Operation**, make the following settings:
 - To boot from flash memory, turn on **Do you want to Boot from flash?**
 - Turn off **Do you want the processor to be held in reset after configuration?**
- 6. Specify the following settings and click **Next**:
 - Turn on **Do you want to use the STRIPE-TO-PLD bridge (Master Port)?**



The stripe-to-PLD bridge signals are enabled to allow the processor to access the slave peripheral in the PLD.

- Under **Interrupts**: turn off both options
 - Under **Trace/Debug**: turn off both options
7. Specify the following settings and click **Next**:

- **External clock reference:** 25 MHz
- **Bypass PLL1:** turn off
- **Desired AHB1 frequency:** 160 MHz
- **AHB2 frequency:** 80 MHz
- **SDRAM Frequency:** 133 MHz



The EPXA1 board uses a 25-MHz oscillator for the reference clock. The AHB1 clock setting is for a –2 speed grade of the EPXA1 device.

8. Specify the following settings and click **Next**:
 - **Registers:** 7FFFC000 address, 16K size
 - **SRAM0:** 20000000 address, 16K size
 - **SRAM1:** 20004000 address, 16K size
 - **SDRAM0:** 00000000 address, 64M size
 - **EBI0 (FLASH):** 40000000 address, 4M size, 8 Wait cycles, Low CS polarity, 16-bit Data Width, 1 Bus clock divide
 - **PLD0:** 80000000 address, 16K size



When you enter the EBI0(FLASH) address, the remaining EBI0(FLASH) settings appear. When you enter the SDRAM0 address, the remaining SDRAM0 settings appear. Choose Micron MT48LC16M16A2 as the SDRAM device, and ensure that the SDRAM port width is set to 16 bits .

9. Click **Finish** to create the software header files and the Verilog HDL instantiation of the stripe.

Compile the Hardware Design

The example design has already been pre-built. To become familiar with the process, or to modify it for your own designs, perform the following steps to compile the hardware design:

1. Choose **Settings** (Assignments menu).
2. Under Category, expand Compiler Settings and choose **Device**.
3. Choose the following settings:
 - Family: **Excalibur ARM**
 - Target device: **Specific device selected**
 - Available devices: **EPXA1F484C1**
4. Click **Device & Pin Options**.
5. Click the **General** tab and choose the following:
 - **Enable INIT_DONE output:** turn on

6. Click the **Configuration** tab and choose **Boot from Flash** from the **Configuration scheme** drop-down list.



Refer to the *Configuration Logic* section of the *Excalibur Devices Hardware Reference Manual* for more information on configuring the PLD.

7. Click the **Unused Pins** tab and select **As inputs, tri-stated** under **Reserve all unused pins**.

This sets all the critical control lines for the interfaces on the EPXA1 board to be pulled to their inactive state.

8. Click **OK** twice.
9. Choose **Start Compilation** (Processing menu) to compile (i.e. synthesize, place-and-route) the hardware design.

This section describes how to specify the software build settings for the Quartus II development tools to compile the software application. The location of the software toolset needs to be specified before you can compile your software application.

The example design has the software build settings already set. To become familiar with the process, or to modify it for your own designs, follow the procedure to set the various compiler, assembler, and linker options.



Before you perform a software build using GNUPro on UNIX workstations (e.g., Linux, Solaris, or HP-UX), make three additional variable settings. The variable settings for Solaris 2.5 are:

```
ALTERA_ARM9GP_HOST = H-sparc-sun-solaris2.5
ALTERA_ARM9GP_ROOT = /edatools/solaris/arm_gnupro
ALTERA_ARM9GP_VER = arm9-020528
```

The variable settings assume the path to the installed GNUPro toolset is:

```
/edatools/solaris/arm_gnupro/arm9-020528/H-sparc-sun-solaris2.5/bin
```

Specifying the Toolset Directory

The following steps specify the directory for the software toolset:

1. Choose **Settings** (Assignments menu).

Compile the Software Application

2. Under **Category**, expand **Files & Directories** and choose **Toolset Directories**.
3. Perform one of the following actions, depending on the software toolset you are using:
 - Click on **GNUPro for ARM** under **Software toolset**, and browse to the GNUPro executable directory *<GNUPro Tools install directory>\bin*.
 - Click on **ADS Standard Tools** under **Software toolset** and browse to the ADS executable directory *<ADS 1.1 install dir>\bin*.
4. Click **OK**.

Software Build Settings for GNUPro

For the example design you will produce two versions of the software build settings for GNUPro: debug settings and release settings. For the debug settings version you will not apply optimization, but include debug information to facilitate using the GNU Insight debugger. You will apply optimization to the release settings version, but debug information is unnecessary because the release settings version will only be used when the software code has been debugged and is known to be working. The release settings version of the software build produces a software image that is faster than the software image produced with debug settings.

The following steps add the software files to the project:

1. Choose **Settings** (Assignments menu).
2. Under **Category**, expand **Files & Directories** and choose **Add/Remove**.
3. Browse to and add the following files:
 - `..\common\main.c`
 - `..\common\uartcomm.c`
 - `..\common\irq.c`
 - `..\common\exceptions.c`
 - `software\crt0.s`
 - `software\epxa1.c`
4. Click **OK**.

Debug Settings for GNUPro

Follow the steps below to specify software build settings that generate the hello world software program with extra debug information.

1. Choose **Settings** (Assignments menu).
2. Under Category, expand Software Build Settings and choose **General**.
3. Choose **Debug** from the **Current Software Build Settings** drop-down list.
4. Choose **CPU** from the Software Build Settings list and specify the following:
 - **Processor architecture:** ARM922T
 - **Software toolset:** GNUPro for ARM
 - **Byte order:** Little endian
 - **Output file format:** Hexadecimal File
 - **Output file name:** Debug\hello.hex
 - Select **Flash memory configuration** under **Programming File Generation** and browse to **hello.sbi**
5. Choose **C/C++ Compiler** from the Software Build Settings list and specify the following:
 - **Level:** Zero
 - **Preprocessor definitions:** type `DEBUG`
 - **Additional include directories:** type `.,..\common`
 - **Generate debug information:** turn on

6. Choose **Assembler** from the Software Build Settings list and specify the following:
 - **Additional include directories:** type .
 - **Generate debug information:** turn on
 - **Keep local symbols in symbol table:** turn on
 - **Use C preprocessor:** turn on

7. Choose **Linker** from the Software Build Settings list and specify the following:
 - **Object/library modules:** type
%ALTERA_ARM9GP_ROOT%/ALTERA_ARM9GP_VER%/ALTERA_ARM9GP_HOST%/lib/gcc-lib/arm-elf/2.96-%ALTERA_ARM9GP_VER%/crtbegin.o,
%ALTERA_ARM9GP_ROOT%/ALTERA_ARM9GP_VER%/ALTERA_ARM9GP_HOST%/lib/gcc-lib/arm-elf/2.96-%ALTERA_ARM9GP_VER%/crtend.o
 - **Additional library directories:** type
%ALTERA_ARM9GP_ROOT%/ALTERA_ARM9GP_VER%/ALTERA_ARM9GP_HOST%/lib/gcc-lib/arm-elf/2.96-%ALTERA_ARM9GP_VER%,
%ALTERA_ARM9GP_ROOT%/ALTERA_ARM9GP_VER%/ALTERA_ARM9GP_HOST%/arm-elf/lib
 - Under **Link type:** select **Custom link script** and browse to **software\armelf.x**
 - **Command-line options:** add -lc -lgcc -lc



The **crtbegin** and **crtend** files set up the environment and clean up for the C run-time libraries.

The linker script file implements the functions required for the standard library calls this program uses.

8. Click **OK**.

9. Click on **Start Software Build** (Processing menu) to build the software application and create a flash programming file.



As an alternative to using the Quartus II software to build the software application, you can run the provided makefile by typing `make debug` at a DOS prompt in the **gnu** directory.

Release Settings for GNUPro

Follow the steps below to specify software build settings that will generate the hello world software program with optimization turned on.

1. Choose **Settings** (Assignments menu).
2. Under Category, expand Software Build Settings and choose **General**.
3. Choose **Release** from the **Current Software Build settings** drop-down list.
4. Click on **CPU** tab folder and specify the following:
 - **Processor architecture:** ARM922T
 - **Software toolset:** GNUPro for ARM
 - **Byte order:** Little endian
 - **Output file format:** Hexadecimal File
 - **Output file name:** Release\hello.hex
 - Select **Flash memory configuration** under **Programming File Generation** and browse to **hello.sbi**
5. Choose **C/C++ Compiler** from the Software Build Settings list and specify the following:
 - **Optimization:** High
 - **Additional include directories:** type . , . . \common
 - **Generate debug information:** turn off
6. Choose **Assembler** from the Software Build Settings list and specify the following:
 - **Additional include directories:** type .
 - **Generate debug information:** turn off
 - **Keep local symbols in symbol table:** turn off
 - **Use C preprocessor:** turn on

7. Choose **Linker** from the Software Build Settings list and specify the following:
 - **Object/library modules:** type
%ALTERA_ARM9GP_ROOT%/ALTERA_ARM9GP_VER%/ALTERA_ARM9GP_HOST%/lib/gcc-lib/arm-elf/2.96-%ALTERA_ARM9GP_VER%/crtbegin.o,
%ALTERA_ARM9GP_ROOT%/ALTERA_ARM9GP_VER%/ALTERA_ARM9GP_HOST%/lib/gcc-lib/arm-elf/2.96-%ALTERA_ARM9GP_VER%/crtend.o
 - **Additional library directories:** type
%ALTERA_ARM9GP_ROOT%/ALTERA_ARM9GP_VER%/ALTERA_ARM9GP_HOST%/lib/gcc-lib/arm-elf/2.96-%ALTERA_ARM9GP_VER%,
%ALTERA_ARM9GP_ROOT%/ALTERA_ARM9GP_VER%/ALTERA_ARM9GP_HOST%/arm-elf/lib
 - Under **Link type:** select **Custom link script** and browse to **software\armelf.x**
 - **Command-line options:** type `-lc -lgcc -lc`
8. Click **OK**.
9. Click on **Start Software Build** (Processing menu) to build the software application and create a flash programming file.



As an alternative to using the Quartus II software to build the software application, you can run the provided makefile by typing `make release` at a DOS-prompt in the **gnu** directory.

Software Build Settings for ADS

For the example design you will produce two versions of the software build settings for ADS: debug settings and release settings. For the debug settings version you will not apply optimization, but include debug information to facilitate using the ADS AXD debugger. You will apply optimization to the release settings version, but debug information is unnecessary because the release settings version will only be used when the software code has been debugged and is known to be working. The release settings version of the software build produces a software image that is faster than the software image produced with debug settings.

The following steps add the software files to the project:

1. Choose **Settings** (Assignments menu).

2. Under Category, expand Files & Directories and choose **Add/Remove**.
3. Browse to and add the following files:
 - `..\common\main.c`
 - `..\common\uartcomm.c`
 - `..\common\irq.c`
 - `..\common\exceptions.c`
 - `software\armc_startup.s`
 - `software\retarget.c`
4. Click **OK**.

Debug Settings for ADS

Follow the steps below to specify software build settings that generate the software program with extra debug information.

1. Choose **Settings** (Assignments menu).
2. Under Category, expand Software Build Settings and choose **General**.
3. Choose **Debug** from the **Current Software Build Settings** drop-down list.
4. Choose **CPU** from the Software Build Settings list and specify the following settings:
 - **Processor architecture:** ARM922T
 - **Software toolset:** ADS Standard Tools
 - **Byte order:** Little endian
 - **Output file format:** Hexadecimal File
 - **Output file name:** Debug\hello.hex (hexadecimal file)
 - Select **Flash File memory configuration** under **Programming File Generation** and browse to **hello.sbi**



A **.sbi** file is a slave binary image file. **hello.sbi** contains the PLD configuration data for the design, which is used in boot-from-flash mode



To generate a programming file, but not include the hardware image, select **Flash memory configuration**, but leave the filename blank.

5. Choose **C/C++ Compiler** from the Software Build Settings list and specify the following settings:
 - **Level:** Low
 - **Goal:** Minimize size
 - **Preprocessor definitions:** type DEBUG
 - **Additional include directories:** type ., . . . \common
 - **Generate debug information:** turn on
6. Choose **Assembler** from the Software Build Settings list and specify the following settings:
 - **Additional include directories:** type .
 - **Generate debug information:** turn on
 - **Keep local symbols in symbol table:** turn on

7. Choose **Linker** from the Software Build Settings list and specify the following settings:
 - Under **Link type**: select **Simple**
 - **Entry symbol name/address**: turn on and type 0
 - **Read-only base address**: turn on and type 0
 - **Read/write base address**: turn on and type 0x20000
 - **Command-line options**: add
-first armc_startup.o(init)



This command-line option sets section `init` in `armc_startup.s` to the reset vector address 0.

8. Click **Apply** and **OK**.
9. Choose **Start Software Build** (Processing menu) to build the software program and create a flash programming file.



As an alternative to using the Quartus II software to build the software application, you can run the provided makefile by typing `make debug` at a DOS prompt in the `ads` directory.

Release Settings for ADS

Follow the steps below to specify software build settings that will generate the **hello world** software program with optimization turned on.

1. Choose **Settings** (Assignments menu).
2. Under Category, expand Software Build Settings and choose **General**.
3. Choose **Release** from the **Current Software Build settings** drop-down list.
4. Choose the **CPU** from the Software Build Settings list and specify the following:
 - **Processor architecture**: ARM922T
 - **Software toolset**: ADS Standard Tools
 - **Byte order**: Little endian
 - **Output file format**: Hexadecimal File
 - **Output file name**: Release\hello.hex
 - Select **Flash memory configuration** under **Programming File Generation** and browse to `hello.sbi`
5. Choose the **C/C++ Compiler** from the Software Build Settings list and specify the following:

- **Optimization:** High
 - **Goal:** Minimize size
 - **Additional include directories:** type `., . . . \common`
 - **Generate debug information:** turn off
6. Choose the **Assembler** from the Software Build Settings list and specify the following:
 - **Additional include directories:** type `.`
 - **Generate debug information:** turn off
 - **Keep local symbols in symbol table:** turn off
 7. Choose the **Linker** from the Software Build Settings list and specify the following:
 - Under **Link type:** select **Simple**
 - **Entry symbol name/address:** turn on and type `0`
 - **Read-only base address:** turn on and type `0`
 - **Read/write base address:** turn on and type `0x20000`
 - **Command-line options:** add `-first armc_startup.o(init)`
 8. Click **Apply** and **OK**.
 9. Choose **Start Software Build** (Processing menu) to build the software program and create a flash programming file.



As an alternative to using the Quartus II software to build the software application, you can run the provided makefile by typing `make release` at a DOS prompt in the `ads` directory.

Configure the Development Board

The EPXA1 device can be configured by a variety of different methods. The configuration process involves setting up the embedded stripe registers and the on-chip SRAM in order to boot, in addition to initializing the PLD array. The processor boots from address `0H`; there are two ways to make code available at this address: boot-from- passive-serial or boot-from-flash.

Boot-from-Passive-Serial Mode

The `BOOT_FLASH` pin is tied high on the EPXA1 development board, but `boot_from_serial` configuration is still possible using the Quartus II programmer. This is because JTAG configuration overrides other modes of configuration. For information on configuring the EPXA1 device using the Quartus II programmer, see the Quartus II on-line help.



Additional steps are required to use the SDRAM in boot-from-serial mode. The information is beyond the scope of this document, so to use boot-from-serial mode with the SDRAM, refer to AN141: *Using the SDRAM Controller*.

Boot-from-Flash Mode

Setting the `BOOT_FLASH` external pin to high puts the device in boot-from-flash mode upon power up. In this mode, the processor accesses the bootcode from flash memory connected to EBIO, where the bottom 32 Kbytes are mapped from address 0H. The stripe registers are mapped to their default addresses, based at 7FFFC000H and the remaining peripherals are not mapped. The `BOOT_FLASH` pin is permanently tied high on the EPXA1 development board.

Altera provides a bootcode program with the Excalibur utilities to facilitate booting from external flash memory. The bootcode performs the following functions:

1. Initializes the device registers and sets up the memory map according to the system build descriptor (`.sbd`) file produced by the MegaWizard Plug-In.
2. Loads the software into RAM whether it is on- or off-chip memory.
3. Resets the watchdog timer and sets the embedded processor's endianness.
4. Loads the PLD configuration data into the device.
5. Passes control to the user's code.

The `hello_flash.hex` flash programming file created in “[Compile the Software Application](#)” on page 17 is used to configure the embedded processor PLD by loading the software and hardware images from an external flash device. The software image is the `hello.hex` (Hexadecimal) file created in “[Compile the Software Application](#)” on page 17 and the hardware image is the `hello.sbi` created in the “[Compile the Hardware Design](#)” on page 16.

Connections and Jumper Settings

To set up the EPXA1 development board in boot-from-flash mode, perform the following steps:

1. Connect one end of the 25-pin parallel cable to the parallel port of the computer and the other end to the ByteBlasterMV download cable.
2. Connect the ByteBlasterMV download cable to the 10-pin JTAG header on the edge of the development board.



If you have not previously used a ByteBlasterMV download cable with your PC, you must install the ByteBlaster driver before continuing. See [“Appendix A—Installing the ByteBlaster Driver”](#) on page 33.

3. Set the jumpers according to [Table 2](#) below.

Jumper	Setting	Description
JSELECT	2-3	Connects both the PLD TAP controller and the processor TAP controller to the ByteBlaster header
CLKA Select	1-2	Selects the on-board oscillator as the source for CLKA
CLKB Select	1-2	Selects the on-board oscillator as the source for CLKB



Refer to the section “Jumpers” in the [Excalibur EPXA1 Development Board Hardware Reference Manual](#) for more information.

4. Connect the power supply to the power connector on the development board.



On power up, LEDs D12, D13, and D14 near the reset switch in the lower right corner of the board illuminate, which indicates that the board is fully powered.

Altera Flash Programmer

Altera provides a flash programming utility, which initially explores the development board, providing a listing of the setup and the JTAG configuration chain. It then downloads a small application into the EPXA1 device and probes the flash device to determine the required programming algorithm. After programming completes, the message `Flash programmed successfully` appears in the command prompt window. The EPXA1 device can be reset and made to boot from the data programmed into the flash memory using the flash programmer.

To configure the EPXA1 device with the flash programming file, proceed as follows:



Debugger software that interfaces with the EPXA1 PLD via JTAG must not be connected to the processor when downloading a design into flash memory.

1. Open a Command Prompt window and change to the **hello\ads** or **hello\gnu** directory.
2. Run the batch file **prog_hw.bat**.
3. Start up HyperTerminal with the settings: 38400 baud rate, 8 data bits, 1 stop bit, no parity, and no flow-control. You should see the LEDs scrolling and the corresponding value displayed on the terminal window.

Debug the Design

To debug the example design, follow the the instructions below for the appropriate ADS or GNU debugger.

ADS AXD Debugger

The ADS tools include the AXD debugger, which can be used to pause the embedded processor and observe the detailed operation of the design.



For more information on the AXD debugger, refer to the *ARM Developer Suite - Debuggers Guide*.

The Excalibur utilities include the RDI (**Altera_RDI.dll**). The RDI is the standard application interface between ARM processors and ARM-supported debuggers. If your debugger supports the RDI from ARM and the ARM922T processor, you can use your existing debugger with an EPXA1 device, using a ByteBlaster™ cable.

To set up AXD and run the application software, perform the following steps.

1. Start AXD by selecting **Start > Programs > ARM Developer Suite > AXD Debugger** (Start menu).
2. Click on **Load Debug Symbols** (File menu) in the AXD debugger window and browse to **ads\debug\hello.elf**. Click **Open**.



Alternatively, click on **Launch Debugger** (Processing menu) in the Quartus II software window to perform steps 1 and 2 above automatically.

3. Select **Configure Target** (Options menu) in the AXD debugger window.
4. If Altera-RDI is listed as a target, click it to highlight it, then click **OK** to connect to the embedded processor. If Altera-RDI is not listed as a target, you must add it, by performing the following steps:
 - a. Click **Add** in the Choose Target window.
 - b. Browse to the directory in which the Quartus II software is installed.
 - c. Navigate to the *<Quartus Installation Directory>\bin* directory.
 - d. Select **Altera-RDI.dll**. Click **Open**.
 - e. Click on Altera-RDI in the Choose Target window; click **OK** to connect to the embedded processor.
5. Choose **Step** (Execute menu) in the AXD debugger window to execute the first assembly instruction.
6. Click on **Go** (Execute menu) in the AXD debugger window to run the application software continuously until it hits the breakpoint automatically set at function `main()`.
7. Click on **Registers** (Processor Views menu) to display the processor registers.



Right-click on an instruction's line number and choose **Toggle Breakpoint** to set or clear a breakpoint.

8. Click on **Memory** (Processor Views menu) and type an address to display the memory contents.

After successfully debugging the design using the debug settings version of the software, you can run the release settings version.

GNUPro Insight Debugger

The GNUPro Toolkit from Red Hat incorporates the Insight graphical debugger. The debugger interfaces with the GNU debugger stub provided by Altera in the Quartus II software, version 2.2 or later. The Altera-supplied stub runs on a host PC and debugs the code running on the Excalibur device using the JTAG debug module.



Refer to the GNUPro Insight debugger manual for more information on the debug commands.

The following steps explain how to set up the GNUPro Insight debugger and run the application software:

1. In a Command Prompt window, start up the GNU debugger stub by typing:

```
<Quartus Installation Directory>\bin\gdbstub.␣
```

2. Execute `<GNUPro Tools installation directory>\bin\arm-elf-gdb.exe` to start Insight.
3. Click on **Open** (File menu) in the Insight debugger and browse to `gnu\debug\hello.elf`.
4. Choose **Connect To Target** (Run menu) in the Insight Debugger window and specify the following in the dialog box:
 - Target: **Remote/TCP**
 - Port: **9999**
 - Ensure that **Download Program** under **More Options** is not selected.
5. Set a breakpoint by right-clicking on the line number and selecting **Set Breakpoint**.
6. Click on **Run** (Run menu) in the Insight debugger to run the application software.
7. Click on **Registers** (View menu) to display the processor registers.
8. Right-click on an instruction's line number and choose **Set Breakpoint** to set a breakpoint.

9. Click on **Memory** (View menu) to display the memory contents.

After successfully debugging the design using the debug settings version of the software, you can run the release settings version.



Notes:

Introduction

If you are using either Windows NT 4.0 or Windows 2000, the Altera ByteBlaster driver must be installed before you can configure the EPXA1 development board using the flash programmer. This section explains how to install the ByteBlaster driver from the **qutilities** directory, either for Windows NT 4.0 or Windows 2000, depending on which operating system you are using.

Installing the ByteBlaster Driver on a Windows NT System

Follow the steps below to install the Altera ByteBlaster driver on a Windows NT 4.0 system:

1. Open the control panel.
2. Select **Multimedia**.
3. Select the **Devices** tab.
4. Select **Add**.
5. Select **Unlisted or Updated Driver** from the **List of Drivers** list box and choose **OK**.
6. If the Quartus II software is installed, type or browse to \<path to Quartus II>\drivers in the text box and choose **OK**.



If the Quartus II software is not installed, but the **qutilities** package is installed, type or browse to \<path to qutilities>\drivers in the text box and choose **OK**.



If neither the Quartus II software nor the **qutilities** package is installed, **utilities** must be installed before installing the ByteBlaster driver.

7. Choose **OK** again in the **Install Driver** window.
8. Select **Altera ByteBlaster** in the **Add Unlisted or Updated Driver** window and choose **OK**.
9. Restart the PC.

Installing the ByteBlaster Driver on a Windows 2000 System

Follow the steps below to install the Altera ByteBlaster driver on a Windows 2000 system:

1. Open the control panel.
2. Select **Add/Remove Hardware** to start the Add/Remove Hardware Wizard and click **Next** to continue.
3. In the Choose a Hardware Task panel, select **Add/Troubleshoot a device** and click **Next** to continue. Windows 2000 searches for new plug and play hardware (**New Hardware Detection** window).
4. In the Choose a Hardware Device window, select **Add a new device** and click **Next** to continue.
5. In the Find New Hardware window, select **No, I want to select the hardware from a list** and click **Next** to continue.
6. In the Hardware Type window, select **Sound, video and game controllers** and click **Next** to continue.
7. In the Select a Device Driver window, select **Have Disk ...**
8. If the Quartus II software is installed, type or Browse to `\<path to Quartus II>\drivers\win2000` and click **OK**.



If the Quartus II software is not installed, but the **qutilities** package is installed, type or Browse to `\<path to qutilities>\drivers\win2000` and click **OK**.



If neither the Quartus II software nor the **qutilities** package is installed, **qutilities** must be installed before installing the ByteBlaster driver.

9. In the Digital Signature Not Found warning dialog box, click **Yes** to continue the installation.
10. In the Select a Device Driver window, select the hardware to install and click **Next** to continue.
11. The Start Hardware Installation window displays the hardware being installed. Click **Next** to continue.
12. In the Digital Signature Not Found warning dialog box, click **Yes** to continue the installation.

13. In the Completing the Add/Remove Hardware Wizard window, click **Finish**. A system dialog appears prompting a reboot so that the new settings can take effect.

Confirming the Installation

After the system is rebooted, follow the steps below to ensure the ByteBlaster driver is accessible:

1. Connect the ByteBlasterMV cable to the PC's parallel port LPT1.
2. Open a command prompt window.
3. Type `jtagconfig`↵ at the command prompt.
4. If the PC cannot connect to the ByteBlasterMV cable, the following message appears:

```
Unable to lock chain (Hardware not attached)
```

5. Type `jtagconfig --add ByteBlaster LPT1`↵.

You can now use the ByteBlasterMV cable to configure the EPXA1 development board.



Notes: